P07- Pruebas de aceptación: Selenium IDE

Os recordamos que el plazo para entregar los ejercicios de esta práctica (P07) termina justo ANTES de comenzar la siguiente sesión de prácticas (P08) en el laboratorio (los grupos de los lunes, por lo tanto, el próximo lunes, los de los martes, el próximo martes, ...). Pasado este plazo los profesores no revisarán vuestro trabajo sobre P07 (de forma individual), con independencia de que se indiquen en clase las soluciones que os permitirán saber si las vuestras son correctas.

Pruebas de aceptación de aplicaciones Web

El objetivo de esta práctica es automatizar pruebas de aceptación de pruebas emergentes funcionales sobre una aplicación Web, para lo que utilizaremos la herramienta Selenium IDE, desde el navegador Chrome.

Tanto para pruebas de aceptación, como para pruebas del sistema, los datos de entrada de nuestros casos de prueba están formados por una **secuencia de entradas** al sistema (eventos del sistema). Dado que vamos a realizar pruebas sobre una aplicación web, cada dato de entrada de nuestra tabla de casos de prueba es del tipo "teclear S", "pulsar con botón derecho sobre Y", "hacer doble click sobre K",... y que el resultado esperado viene dado por la **secuencia de acciones** que se desencadenan al tener lugar cada uno de los eventos: por ejemplo: cargar la página X, enviar el texto tecleado al servidor, mostrar por pantalla el valor X,... Es decir, cambia algo el "aspecto" de la tabla, por así decirlo, pero sigue siendo una tabla de casos de prueba (conjunto de entradas concretas, y resultado esperado (también concreto).

Otra observación más: puesto que el resultado esperado no es un único "valor", sino que está formado por la secuencia de salidas obtenidas de todas y cada una de las acciones que se desencadenan al producirse los eventos, nuestro *driver* normalmente tendrá varios "*asserts*" (por ejemplo, lo habitual es que cada vez que se cargue una nueva página (o parte de ella, si utilizamos ajax) verifiquemos que los datos que se han cargado y se han mostrado en el navegador son los correctos.

Implementaremos los *drivers* para nuestras pruebas con *scripts* de comandos Selenese (que posteriormente guardaremos en formato json). Tal y como hemos visto en clase, Selenium IDE nos permite "capturar" las acciones del usuario desde el navegador para generar de forma automática las instrucciones (código) de nuestros *drivers*, los cuales pueden ejecutarse de forma automática tantas veces como sea necesario. Esta forma de uso de la herramienta ("grabando" las acciones del usuario sobre el navegador) es lo que se conoce como el patrón *record-playback*.

Recuerda también que, aunque la práctica sea muy "guiada", debes tener claro lo que estás haciendo en cada momento.

Bitbucket

El trabajo de esta sesión también debes subirlo a *Bitbucket*. Todo el trabajo de esta práctica deberá estar en el directorio **P07-Selenium-IDE** dentro de tu espacio de trabajo, es decir, dentro de tu carpeta: ppss-2021-Gx-apellido1-apellido2.

Navegador Chrome y Selenium IDE

En la máquina virtual tenéis instalado el navegador Firefox con la extensión Selenium IDE. Hemos constatado que Selenium IDE + Firefox no funcionan bien en la máquina virtual (sin embargo Firefox+Selenium IDE, con las mismas versiones usadas en la máquina virtual pero bajo *macOS* funcionan perfectamente).

Vamos a instalar Chrome en la máquina virtual, puesto que la combinación Chrome+Selenium IDE no plantea ningún problema.

Desde un terminal, teclea los siguientes cuatro comandos (el símbolo ">" representa el prompt, no hay que copiarlo):

```
> wget -q -O - https://dl.google.com/linux/linux_signing_key.pub | sudo apt-key add -

> sudo sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable
  main" >> /etc/apt/sources.list.d/google-chrome.list'

> sudo apt update

> sudo apt install google-chrome-stable
```

Ejecutamos Chrome desde: ***Inicio → Internet → Google Chrome***

Finalmente instalamos Selenium IDE como una extensión del navegador Chrome, desde.
https://chrome.google.com/webstore/detail/selenium-ide/mooikfkahbdckldjjndioackbalphokd

Si quieres acceder a Chrome desde un icono en el escritorio haz lo siguiente (desde el terminal):

(1) Primer copiamos el lanzador en el escritorio con:

```
cp /usr/share/applications/google-chrome.desktop $HOME/Desktop/
```

(2) A continuación, desde el menú contextual del icono copiado en el escritorio marcamos "Confiar en este ejecutable".:

Aplicación web para los ejercicios de esta sesión

Utilizaremos una aplicación Web Java denominada ***JPetStore*** (http://mybatis.github.io/spring/sample.html) sobre la que haremos las pruebas. Se trata de una versión simplificada de la demo de Sun denominada *Java PetStore*.

La aplicación a probar la tenéis descargada en vuestro *$HOME* en la máquina virtual, en la carpeta ***jpetstore-6***.

Nota: se podría descargar con el siguiente comando:

```
>git clone https://github.com/mybatis/jpetstore-6.git
```

Para familiarizarte con el código puedes abrir el proyecto maven "jpetstore-6" desde IntelliJ. El directorio de fuentes está estructurado en 4 paquetes:

- **domain**: contiene las clases que representan los objetos del dominio del negocio, con sus getters y setters,
- **mapper**: contiene las interfaces para “mapear” los objetos java con los datos persistentes,
- **service**: contiene las clases con la lógica de negocio de la aplicación,
- **web.actions**: contiene las acciones, es decir, la lógica de la capa de presentación de la aplicación.

La aplicación utiliza el patrón MVC (Modelo-Vista-Controlador) con tres capas. (Puedes consultar http://www.mybatis.org/jpetstore-6/ para una descripción más detallada):

- **vista** (capa de presentación): formada por ficheros jsp y ActionBeans,
- **controlador** (capa de negocio): formada por objetos del dominio y servicios, y
- **modelo** (capa de datos): formada por interfaces de mapeo con datos persistentes

En carpeta jpetstore-6 el código no es exactamente el original, ya que hemos hecho las siguientes modificaciones:

- **fichero jpetstore.css**: En la línea 174 hemos borrado dos llaves que no tienen nada en su interior.
- **fichero AccountActionBean.java:** hemos cambiado la línea 119, "authenticated=true", por "authenticated=false"

Construcción y despliegue de la aplicación web

Las aplicaciones web se empaquetan en ficheros **.war,** y dicho artefacto tiene que ser desplegado en un servidor (un servidor web o un servidor de aplicaciones). Una vez que nuestra aplicación web esté en ejecución en el servidor, podemos acceder a ella a través del navegador, utilizando la url en la que ha sido desplegada nuestra aplicación.

Maven Build profiles

(se ejecutan con: `mvn –P profileID`)

El pom de nuestra aplicación web contiene varios "*build profiles*" (etiquetas *<profile>,* anidadas en la etiqueta *<profiles>*). Maven usa los "***build profile***" com una forma de asegurar la "portabilidad" de nuestras construcciones. Un *build profile* usa los elementos definidos en el pom, y puede añadir nuevos elementos y/o modificar los existentes.

Por ejemplo, en el pom usamos el plugin "***cargo***" para poner en marcha un servidor web, concretamente "tomcat" y desplegar ahí nuestra aplicación. Si queremos cambiar el contenedor para desplegar nuestra aplicación web, podemos usar uno de los "profiles" creados, o añadir uno nuevo. Dado que en la máquina virtual, en la carpeta $HOME/descargas tenéis un *zip* con el servidor de aplicaciones ***wildfly 21.0.1***, vamos a usar este contenedor para desplegar nuestra aplicación web, y vamos a añadir un nuevo profile a los ya existentes en nuestro pom. Concretamente añadiremos el *build profile* siguiente::

```
<profile>
  <id>wildfly21</id>
  <properties>
    <wildfly.major-version>21</wildfly.major-version>
    <wildfly.version>21.0.1.Final</wildfly.version>
    <cargo.maven.containerId>wildfly${wildfly.major-version}x</cargo.maven.containerId>
    <cargo.maven.containerUrl>http://download.jboss.org/wildfly/${wildfly.version}/wildfly-
${wildfly.version}.zip</cargo.maven.containerUrl>
  </properties>
</profile>
```

Hemos añadido la siguiente línea en la configuración del plugin ***cargo***. El plugin se descarga el contenedor web elegido en nuestro directorio de descargas (*/home/ppss/Descargas*). Como ya hemos copiado el zip en ese directorio, no será necesario descargarlo, simplemente lo descomprimirá en *target/cargo* cuando construyamos el sistema:

```
<plugin>
   <groupId>org.codehaus.cargo</groupId>
   <artifactId>cargo-maven2-plugin</artifactId>
   <version>${cargo-maven2-plugin.version}</version>
     ...
     <zipUrlInstaller>
        <url>${cargo.maven.containerUrl}</url>
        <!-- añadimos la siguiente línea -->
        <downloadDir>${env.HOME}/Descargas</downloadDir>
     </zipUrlInstaller>
     ...
</plugin>
```

Para **empaquetar** y **desplegar** nuestra aplicación web (desde el **TERMINAL**, y la carpeta *$HOME/jpetstore-6*), usaremos el comando:

> `mvn package cargo:run –P wildfly21` **(1)**

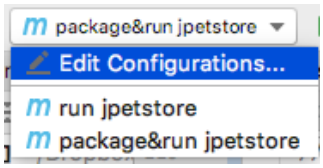
Dado que no vamos a editar nuestra aplicación web, las siguientes veces que necesitemos ejecutar la aplicación podemos hacerlo con:

> `mvn cargo:run –P wildfly21` **(2)**

También podemos **empaquetar** y **desplegar** la aplicación desde **INTELLIJ**:

a) Usando elementos de configuración.

Crearemos dos elementos de configuración maven con los nombres "package&run jpetstore" al que asociaremos el comando (1) y "run jpetstore", que tendrá asociado el comando (2)



b) Desde la ventana Maven de IntelliJ.

Desde el elemento Profiles, desmarcamos el profile *tomcat90* (que es el que está activo por defecto) y marcamos *wildfly21*

A continuación podemos ejecutar la fase *package,* y arrancar y desplegar el war generado a través del plugin *cargo*.

Una vez que hemos arrancado el servidor de aplicaciones y desplegado el *war,* ya puedes **ejecutar** la aplicación web desde el navegador Chrome, accediendo a la URL:

`http://localhost:8080/jpetstore`

Cuando arrancamos wildfly pondremos en marcha dos procesos: la consola de administración de *wildfly* (en el puerto 9990), y el servidor web que contiene nuestra aplicación *jpetstore*, que estará escuchando en el puerto 8080-

Si has arrancado el servidor desde IntelliJ, puedes detenerlo usando el botón **cuadrado rojo** del panel de ejecución.

Si cierras IntelliJ y te has dejado el servidor "en marcha", puedes abortar el proceso posteriormente desde el terminal usando el comando lsof: **> lsof -i:8080**

Este comando nos muestra la información de qué proceso está ocupando ese puerto. Si hay algún proceso escuchando en ese puerto, nos mostrará, entre otras cosas, su PID.

Sabiendo el PID podemos abortar el proceso con **> kill -9 pidProceso**

NOTA:Dado que IntelliJ consume bastantes recursos de la máquina y no vamos a necesitar editar el código de nuestra aplicación web, os aconsejamos que en sucesivas ejecuciones uséis directamente el terminal. Y ejecutéis el comando (2) anterior desde la carpeta j*petstore-6* (es decir, desde el directorio en el que se encuentra el pom del proyecto maven). Para parar el servidor puedes detener el proceso pulsando *Ctrl-C* desde el terminal

Ejercicios

Accede a la aplicación web y familiarízate con ella antes de comenzar a crear nuestros tests de aceptación.

La pantalla principal nos muestra la bienvenida a la tienda de animales. Una vez que entres en la tienda puedes mostrar una página de ayuda pulsando sobre el enlace "?" en la parte superior central de la página. Pinchando sobre el logo de la parte superior izquierda vuelves a la página principal de la tienda, en donde encontrarás 5 catálogos de diferentes animales, desde los que podrás seleccionar los que te interesen para añadirlos a tu carrito de la compra y luego poder hacer efectiva dicha compra si eres usuario de la tienda.

Vamos a implementar algunos tests de pruebas de aceptación de propiedades emergentes funcionales con Selenium IDE sobre la aplicación web.

➪ ➪ Ejercicio 1: Primer Test Case Selenium (Caso de prueba 1)

Vamos a realizar una primera **grabación de una secuencia de acciones** para familiarizarnos con Selenium IDE. Abre la herramienta pulsando sobre el icono de la barra de herramientas de Chrome, y selecciona "*Record a new test in a new project*":

- Nombre del proyecto: *DriversSeleniumIDE*.
- Base URL: *http://localhost:8080/jpetstore*
- ... y comenzamos a grabar

Antes de continuar, y para que veas en todo momento lo que está generando la herramienta, redimensiona las ventanas y asegúrate de que son totalmente visibles tanto la de grabación (ventana Chrome con el mensaje "*Selenium IDE is recording*"), como la herramienta Selenium IDE.

El **caso de prueba** consiste en el siguiente escenario:

1. Entra en la tienda
2. Verifica que el título de la página es "JPetStore Demo". Para ello con botón derecho desde cualquier sitio de la página selecciona el comando selenese "*verify Title*" (desde el menú contextual)
3. Haz click sobre la imagen del pez
4. Pulsa sobre el ítem del catálogo con identificador FI-SW-02
5. Verifica que el texto "Toothless Tiger Shark" está presente en la página (comando "*verify Text*")
6. Pulsa sobre “Return to Fish”, seguidamente sobre “Return to Main Menu”, y verifica que el texto "Sign In" está presente en la página

"Graba" con la herramienta el código del driver que implementa el caso de prueba anterior. Observa cómo se van generando automáticamente los comandos selenese a medida que interaccionamos con nuestra aplicación.
Detén la grabación, y pon el nombre "**Caso de prueba 1**" al driver implementado.
La implementación del driver debe quedar así:

Puedes observar que, tal y como hemos explicado en clase, cada comando requiere uno o dos parámetros.
Si el comando requiere un "locator", éste será el primer parámetro. Un “locator” representa un elemento html. Así, por ejemplo, el comando ***click*** tiene como parámetro el locator ***linkText***, asociado a un hiperenlace html.
Observa también que las cadenas de caracteres NO llevan "comillas"

Driver "Caso de prueba 1"			
1	open	/jpetstore/	
2	set window size	873x765	
3	click	linkText=Enter the Store	
4	verify title	JPetStore Demo	
5	click	css=area:nth-child(2)	
6	click	linkText=FI-SW-02	
7	click	css=h2	
8	verify text	css=h2	Tiger Shark
9	click	linkText=Return to FISH	
10	click	linkText=Return to Main Menu	
11	verify text	linkText=Sign In	Sign In

Fíjate también en que Selenium añade alguna acción que realmente no hemos ejecutado: por ejemplo el comando 2 para redimensionar la ventana del navegador. Esta línea la añadirá Selenium en todos nuestros tests. Podríamos borrarla, aunque por comodidad no lo vamos a hacer. También puedes ver que cuando hemos insertado el comando verify, la herramienta también ha grabado el "click" que hemos hecho en la página (comando 7) para abrir el menú contextual y elegir el comando selenese. Vamos a dejarlos porque de hecho, son las acciones que realmente hemos necesitado hacer sobre la página. No los incluimos de forma explícita en los casos de prueba, pero se añaden automáticamente. Puedes probar a "comentar" estos los comandos que *selenese* nos ha añadido y que no son estrictamente los pasos indicados en el escenario de prueba. Para "comentar" un comando usaremos el botón **"//"** a la derecha del campo de texto Command. Este botón "**habilita/deshabilita" la ejecución del comando** correspondiente. En concreto, prueba a comentar los comandos 2 y 7 y verás que el test sigue funcionando correctamente. También puedes borrarlos usando la tecla "del".

Guarda el proyecto en un fichero con nombre "DriversSeleniumIDE" en el directorio *ppss-2021-Gx-.../P07-Selenium-IDE.* El ficheros generado tendrá extensión .side, pero es un fichero de texto en formato JSON.

Ahora ejecuta el test que acabas de crear (segundo icono que tiene la forma de triángulo). Prueba a cambiar la velocidad de ejecución de forma que puedas ver perfectamente la ejecución de todos y cada uno de los comandos selenese de nuestro driver.

➪ ➪ Ejercicio 2: Test Case *compraNewUser*

Vamos a crear un nuevo test en nuestro proyecto Selenium, con el nombre "**test-*compraNewUser***". En este caso vamos a usar un escenario en el que un usuario que todavía NO está registrado en la página quiere hacer una compra de varios productos. En este caso, después de añadir los productos al carrito y proceder a hacer efectiva la compra, ésta no se completará si el usuario no se ha creado una cuenta en la tienda. Después de crear la cuenta, se podrá proceder a la compra de los productos almacenados en el carrito (éstos se mantienen en el carrito durante el proceso de registro del nuevo cliente).

Como ya hemos indicado en este test actuaremos como un usuario no registrado que quiere hacer una compra de dos perros y un gato. Para ello pulsamos el botón de grabación y accedemos a nuestra aplicación desde el navegador (URL: *http://localhost:8080/jpetstore*).

El **escenario** elegido para el **caso de prueba** es el siguiente:

1. Entramos en la tienda y pulsamos con botón izquierdo del ratón sobre la imagen del perro
2. Nos aseguraremos de que estamos en la página correcta verificando que el texto “Dogs” se muestra en la página
3. Vamos a **comprar un Bulldog**. Verificaremos (igual que hemos hecho antes) que el texto “Bulldog” aparece en la página
4. Pulsamos sobre el enlace K9-BD-01 y verificamos que estamos en la página correcta verificando que el texto “Bulldog” que aparece sobre la tabla.
5. Queremos comprar un **macho adulto**. Verificamos que en la página aparece el texto “Male Adult Bulldog”. A continuación pinchamos sobre “Add to cart” de la primera fila de la tabla. Verificamos que estamos en la página correcta comprobando que el texto “Shopping Cart” aparece en la nueva página cargada. Queremos comprar dos unidades, por lo que editaremos el campo de texto y pondremos un 2 (no es necesario que pulsemos “return” después de poner el 2). A continuación seleccionamos la opción “Update Cart” y verificaremos que aparece el texto “Sub Total:$37,00”.
6. Ahora vamos a **comprar un gato**. Para lo cual pinchamos sobre el enlace “Cats” en la parte superior de la página. Verificamos que estamos en la página correcta comprobando que aparece el texto “Cats”. Antes de seguir, vamos a comprobar que efectivamente en el carrito tenemos los 2 bulldogs, para lo cual pinchamos sobre el icono del carrito de la compra de la parte superior de la página. Verificamos que efectivamente aparece el texto “Male Adult Bulldog” y el sub total es el mismo de antes. Volvemos de nuevo al enlace “Cats”, y procedemos a comprar un ejemplar de persa. Para ello verificamos el texto “Persian”, y pinchamos sobre FL-DLH-02. Verifica que el texto “Adult Female Persian” está presente en la página y añade al carrito una hembra adulta. Ahora nuestro carrito debe contener los productos “Male Adult Bulldog” y “Adult Female Persian”, y el nuevo sub total debe ser de 130,50 dólares.
7. Antes de proceder a hacer efectiva la compra, pensamos que igual no es buena idea la compra de dos perros y un gato. **Decidimos no comprar el gato**, por lo que eliminamos este ítem. Ahora vamos a comprobar que el identificador EST-16 NO aparece en la página. Esto no lo podemos hacer de forma automática, sino que tendremos que introducir el comando manualmente. Se trata del comando “*verify not text*”. No detengas la grabación.
8. Selecciona la última línea del editor de texto, que está vacía y aparece sombreada en azul en su parte superior. A continuación, en el campo "Command" busca y selecciona el comando "verify not text". En el campo de texto Value escribiremos EST-16. Para escribir el valor del campo target, pulsamos el botón a la izquierda de la lupa (para activar la búsqueda del elemento en el navegador), nos situamos con el ratón sobre la tabla en el navegador, cuando TODA la tabla aparezca resaltada, pulsamos con el botón izquierdo del ratón, y el cuadro de texto *target* se rellenará con el locator de la tabla.

9. Haz click sobre la siguiente línea en blanco del editor de comandos selenese, y continúa la grabación verificando que en la página el sub total vuelve a ser de 37 dólares. Hasta ahora nuestro script de pruebas debe tener el siguiente "aspecto" (No detengas la grabación):

Driver "CompraNewUser"			
1	open	http://localhost:8080/jpetstore/	
2	set window size	647x497	
3	click	linkText=Enter the Store	
4	click	css=area:nth-child(3)	
5	click	css=tr:nth-child(2) > td:nth-child(2)	
6	verify text	css=tr:nth-child(2) > td:nth-child(2)	Bulldog
7	click	linkText=K9-BD-01	
8	click	css=h2	
9	verify text	css=h2	Bulldog
10	click	css=tr:nth-child(2) > td:nth-child(3)	
11	verify text	css=tr:nth-child(2) > td:nth-child(3)	Male Adult Bulldog
12	click	linkText=Add to Cart	
13	click	css=h2	
14	verify text	css=h2	Shopping Cart
15	click	name=EST-6	
16	type	name=EST-6	2
17	click	name=updateCartQuantities	
18	click	css=tr:nth-child(3) > td:nth-child(1)	
19	verify text	css=tr:nth-child(3) > td:nth-child(1)	Sub Total: $37,00
20	click	css=a:nth-child(7) > img	
21	click	css=h2	
22	verify text	css=h2	Cats
23	click	name=img_cart	
24	click	css=td:nth-child(3)	
25	verify text	css=td:nth-child(3)	Male Adult Bulldog
26	click	css=tr:nth-child(3) > td:nth-child(1)	
27	verify text	css=tr:nth-child(3) > td:nth-child(1)	Sub Total: $37,00
28	click	css=a:nth-child(7) > img	
29	click	css=tr:nth-child(3) > td:nth-child(2)	
30	verify text	css=tr:nth-child(3) > td:nth-child(2)	Persian
31	click	linkText=FL-DLH-02	
32	click	css=tr:nth-child(3) > td:nth-child(3)	
33	verify text	css=tr:nth-child(3) > td:nth-child(3)	Adult Male Persian
34	click	linkText=Add to Cart	
35	click	css=tr:nth-child(2) > td:nth-child(3)	
36	verify text	css=tr:nth-child(2) > td:nth-child(3)	Male Adult Bulldog
37	click	css=tr:nth-child(3) > td:nth-child(3)	
38	verify text	css=tr:nth-child(3) > td:nth-child(3)	Adult Female Persian
39	click	css=tr:nth-child(4) > td:nth-child(1)	
40	verify text	css=tr:nth-child(4) > td:nth-child(1)	Sub Total: $130,50
41	click	css=tr:nth-child(3) .Button	
42	verify not text	css=table	Adult Female Persian
43	verify text	css=tr:nth-child(3) > td:nth-child(1)	Sub Total: $37,00

10. Ahora procederemos a **realizar la compra**, pinchando sobre "Proceed to Checkout". Verificamos que en la nueva página nos aparece el mensaje: "You must sign on before attempting...". Vamos a probar a introducir el login "z" y el password "z". Verificaremos que en la página aparece el texto: "Invalid username or password...". Como somos un usuario nuevo, tendremos que **registrarnos** primero para poder hacer la compra, por lo que pinchamos sobre "Register now". Nos aparecerá una página con el texto "User Information". Rellenamos los campos. Por ejemplo podemos poner como User ID y password el carácter "z". Rellenaremos todos los campos de "Account

Information" con la letra "z" (por ejemplo. Si quieres, puedes poner cualquier cadena de caracteres). Recuerda rellenar TODOS los campos. Dejaremos sin rellenar el apartado "Profile Information" como aparece por defecto. Finalmente pincharemos sobre "Save Account Information".

IMPORTANTE!!: Al introducir el nombre de usuario, se graba como valor de target el locator "id" del campo de texto etiquetado como "User ID" (compruébalo). El valor de este atributo cambiará cada vez que se recargue la página, por lo que vamos a tener que usar otro "locator". Pulsa sobre el desplegable de "Target" y elige el segundo locator de la lista (name=username). De no hacerlo así, si repetimos el test, éste fallará. Puedes hacer los cambios sin detener la grabación. Recuerda que **siempre que introduzcas el valor del campo User ID**, se debe usar el ***locator name*** en lugar del *locator id*.
Selenium IDE, cuando usamos un *locator* en el campo *target*, siempre guarda todos los *locators* alternativos que podríamos usar con ese elemento, por eso nos aparecen más opciones cuando pulsamos la lista desplegable de Target.

11. Después de crear la cuenta verificamos que el texto "Sign In" aparece en la pantalla. Después volvemos a nuestro **carrito de la compra** pulsando sobre el icono del carrito de la parte superior de la pantalla. Verificaremos que aparece el texto "Shopping Cart" y que el subtotal sigue siendo de $37,00. Ahora pulsamos sobre el botón "Proceed to Checkout", **introducimos el login y password "z", "z"** y entramos. Volvemos a seleccionar el carrito de la compra y comprobamos que el subtotal sigue siendo de 37,00 dólares.
12. Volvemos a pulsar sobre "Proceed to checkout", sobre "continue" y confirmamos. A continuación verificaremos que el nombre del usuario de la dirección de facturación es el correcto ("z"), y que la primera dirección de envío es la correcta. ("z"). Finalmente cerramos la sesión pulsando sobre "Sign Out" y accedemos al carrito de la compra para verificar que el subtotal ahora es de 0 euros. A continuación mostramos la secuencia de comandos que hemos añadido (por simplicidad hemos quitado de la secuencia los "clicks" previos a la inserción y verificación de texto):

Driver "CompraNewUser"			
44	click	linkText=Proceed to Checkout	
46	verify text	css=li	You must sign on ...
48	type	name=username	z
50	type	name=password	z
51	click	name=signon	
53	verify text	css=li	Invalid username ...
54	click	linkText=Register Now!	
55	type	name=username	z
57	type	name=password	z
59	type	name=repeatedPassword	z
...	...	...	...
79	type	name=account.country	z
80	click	name=newAccount	
81	verify text	linkText=Sign In	Sign In
82	click	name=img_cart	
84	verify text	css=h2	Shopping Cart
85	click	css=tr:nth-child(3) > td:nth-child(1)	
86	click	name=img_cart	
88	verify text	css=tr:nth-child(3) > td:nth-child(1)	Sub Total: $37,00
89	click	linkText=Proceed to Checkout	
91	type	name=username	z
93	type	name=password	z
94	click	name=signon	
95	click	name=img_cart	
97	verify text	css=tr:nth-child(3) > td:nth-child(1)	Sub Total: $37,00
98	click	linkText=Proceed to Checkout	

Driver "CompraNewUser"			
99	click	name=newOrder	
101	verify text	css=tr:nth-child(3) > td:nth-child(2)	z
103	verify text	css=tr:nth-child(5) > td:nth-child(2)	z
105	verify text	css=tr:nth-child(12) > td:nth-child(2)	z
106	click	linkText=Sign Out	
107	click	name=img_cart	
109	verify text	css=tr:nth-child(3) > td:nth-child(1)	Sub Total: $0,00

Antes de ejecutar el test tendremos que "borrar" el usuario que hemos creado durante la creación del script de pruebas (driver). Puesto que la aplicación utiliza una base de datos en memoria (HSQLDB), una forma de "limpiar" la base de datos es reiniciar la aplicación. Para ello bastaría con parar y volver a arrancar el servidor de aplicaciones Wildfly. Podemos hacerlo desde IntelliJ o desde un terminal.

Una vez que volvamos a arrancar el servidor, ya podemos ejecutar nuestro test desde Selenium IDE. Puedes poner la velocidad más baja de ejecución para poder ir viendo claramente los pasos (también puedes pausar la ejecución en cualquier momento).

Si intentas repetir el test sin reiniciar el servidor puedes comprobar que el test fallará puesto que estaremos intentando darnos de alta dos veces en el sistema.

➪ ➪ Ejercicio 3: Test Case *compraOldUser*

Ahora vamos a crear otro script de prueba (driver) con el nombre "**test-compra-old-user**". En este caso se trata de un usuario previamente registrado, por ejemplo, el usuario z que acabamos de crear, que quiere comprar 500 ejemplares de iguana verde adulta (el identificador del producto es: RP-LI-02, y el identificador del ítem es: EST-13). Se trata de comprobar que, después de hacer la compra, deben quedar 500 ejemplares menos en stock.

Para ello tendrás que incluir en el script de pruebas el acceso a la página en la que se muestra la información del stock inicial de un ejemplar, recordar esta cantidad para luego restarle 500 unidades, y volver a navegar por dicha página para comprobar que efectivamente el nuevo valor ha disminuido en 500 unidades. Para poder programar esto, vamos a tener que utilizar **variables** para almacenar los valores que necesitemos y operar con ellos, ya que vamos a trabajar con contenidos de la página html que son dinámicos (no sabemos a priori cuál será el stock, y dependiendo de la cantidad que queramos comprar, este stock tendrá un valor que tampoco conocemos a priori). En los siguientes apartados indicamos que comandos podéis utilizar para trabajar con la información dinámica que genera la página.

Nuestro **caso de prueba** consistirá en el siguiente **escenario** (puedes grabar parte de él, y más adelante explicamos cómo programar los pasos 5 y 9. Recuerda que en cualquier momento podemos añadir/modificar/borrar cualquier comando selenese del editor):

1. Entramos en la tienda y nos logueamos como el usuario 'z' con password 'z'. Verificamos que estamos en la página correcta (debe aparecer el texto "Welcome z").
2. Pulsamos con botón izquierdo del ratón sobre la imagen de los reptiles
3. Seleccionamos el producto asociado a la iguana
4. Accedemos al identificador del item (EST-13)
5. (Explicaremos este paso a continuación). Guardamos el texto con la información número de ejemplares en stock en una variable. Extraemos de dicha cadena de caracteres solamente el número de ejemplares disponibles, y almacenamos dicho valor en una segunda variable.
6. Añadimos al carrito el producto seleccionado (Iguana verde adulta)
7. Cambiamos la cantidad a comprar, ya que deseamos comprar 500 unidades y actualizamos el carrito
8. Procedemos a realizar la compra (Botón "Proceed to Checkout"), y confirmamos los datos.

9. Volvemos a acceder a reptiles, a continuación a la iguana, y posteriormente a la iguana verde adulta. Verificamos que la cantidad de ejemplares restantes es la correcta (tiene que haber 500 unidades menos disponibles que antes de realizar la compra)
10. Cerramos la sesión (y verificamos que en la página aparece el texto "*Sign In*").

Para programar el paso 5 necesitas usar el comando Selenese ***store text***. Dicho comando almacena el texto de un elemento de nuestra página en una variable, por lo tanto requiere dos parámetros. Puedes insertar el comando "store text" y activar el localizador de elementos del campo "Target" para buscar el valor del locator adecuado. Podemos usar también el comando ***echo*** para "imprimir" en la pantalla de *Log* el valor de la variable del comando anterior. Para referenciar el valor de una variable, pongamos por ejemplo var1 se utiliza el nombre de la variable entre llaves y precedida de un "$", es decir ${var1}.

Nota: el comando store text hay que introducirlo manualmente

store text	css=tr:nth-child(5) > td	stockText
echo	Texto con el stock de iguanas	
echo	${stockText}	

La variable *stockText* contiene el texto "10000 in stock", pero necesitamos almacenar solamente la cantidad (10000), en lugar de todo el texto, para ello puedes utilizar el comando Selenese ***execute script***. Como ya hemos indicado en clase, este comando ejecuta un *script Javascript*. Para extraer un valor numérico de una cadena de caracteres podemos usar el siguiente *script*:

```
var res = String(${stockText}).match(/\d+\.?\d*/g);
return res;
```

La función *String* convierte un valor (en este caso el valor de nuestra variable stockText) en una cadena de caracteres. A continuación aplicamos sobre dicho string la funcion *match()*, que devuelve la subcadena que coincida con el patrón (expresión regular) que se pasa como parámetro. Finalmente usamos la sentencia *return* para devolver el resultado final.

execute script	var res = String(${stockText}).match(/\d+\.?\d*/g); return res;	stockNumber
echo	Valor de stock (solo numeros)	
echo	${stockNumber}	

Usaremos una tercera variable para almacenar la diferencia entre el valor del stock inicial (stockNumber) y las 500 unidades que queremos comprar. Verificaremos que el valor resultante de la variable es el correcto (es decir, que ahora hay 500 unidades menos en stock de las que había antes de realizar la compra). Para obtener la diferencia podemos usar de nuevo el comando *execute script*.

execute script	return ${stockNumber} - 500	stockEsperado
echo	Valor esperado	
echo	${stockEsperado}	

Para verificar que se ha actualizado correctamente el stock después de la compra, usaremos el comando ***assert***. Dicho comando usa como primer parámetro el nombre de una variable (sin llaves), que contiene el valor esperado, y como segundo parámetro el valor que queremos comparar con el esperado.

assert	stockEsperado	${stockNumberUpdated)

En este caso, ${stockNumberUpdated} es el valor numérico del stock que consultamos después de hacer la compra.

➪ ➪ Ejercicio 4: Test Case *controlFlow*

Ahora vamos a crear otro script de prueba (driver) con el nombre "***controlFlow***". La idea es practicar con alguno de los comandos que permiten alterar la secuencia de ejecución del driver. Por ejemplo, el comando ***if***. Este comando requiere un argumento que pueda evaluarse a true o false.

El caso de prueba en consiste en el siguiente **requerimiento**:

1. Entramos en la tienda y después de verificar que aparece el texto "Sign In", pulsamos sobre el enlace e introducimos el login "ppss", y password "ppss".
2. Si el usuario no tiene cuenta en la tienda (más adelante explicamos cómo saber esto), entonces creamos el nuevo usuario, con login y password "ppss", y puedes rellenar el resto de campos con el valor "a". A continuación nos logueamos en la tienda de nuevo, y ahora nos debe aparecer el mensaje "Welcome a!).
3. Después de entrar con éxito en la tienda, verificamos que en la página aparece el mensaje "Welcome a!" y a continuación saldremos de la tienda (opción "Sign Out").

Para averiguar si el usuario tiene cuenta en la tienda o no, tenemos que usar la información del código de las páginas web a las que somos redirigidos en el caso de que el usuario ya tenga la cuenta creada o no. Si te fijas, ambas páginas son diferentes, y además, no hay ningún elemento (html) de la página que tenga el mismo locator pero diferentes valores en ambas.

En este caso, vamos a usar el comando ***store xpath count.*** Este comando requiere dos parámetros, en el primero indicaremos una ruta relativa, en el segundo almacenaremos el número de elementos de la página que contienen ese *path*. Aprovecharemos el hecho de que la página que muestra cuando el usuario no está registrado contiene 2 formularios, mientras que la página que muestra cuando el usuario ya está registrado y consigue entrar al sistema sólo tiene 1 formulario. La ruta *xpath* que vamos a usar será, por lo tanto, ***xpath=//form.***

A continuación mostramos parte del contenido de tu driver (omitimos alguno de los clicks y pasos que ya hemos explicado antes).

Driver "ControlFlow"		
open	http://localhost:8080/jpetstore/	
set window size	807x869	
click	linkText=Enter the Store	
click	id=MenuContent	
verify text	linkText=Sign In	Sign In
click	linkText=Sign In	
type	name=username	ppss
type	name=password	ppss
click	name=signon	
store xpath count	xpath=//form	formulario
echo	${formulario}	
if	${formulario}==2	
echo	usuario no válido	
click	linkText=Register Now!	
type	name=username	ppss
type	name=password	ppss
type	name=repeatedPassword	ppss
type	name=account.firstName	a
...	...	
click	name=newAccount	
click	linkText=Sign In	
type	name=username	ppss
type	name=password	ppss
click	name=signon	
end		
verify text	id=WelcomeContent	Welcome a!
click	linkText=Sign Out	
verify text	linkText=Sign In	Sign In

En este caso vamos a usar el operador "==" de Javascript.
El operador "===" realiza una igualdad estricta, eso significa que tienen que coincidir los valores y los tipos.

Ejecuta el driver antes de crear el usuario ppss y repite el test para comprobar que funciona en ambos casos.

Resumen

¿Qué conceptos y cuestiones me deben quedar CLAROS después de hacer la práctica?

PRUEBAS DEL SISTEMA Y ACEPTACIÓN

- Las pruebas del sistema son pruebas de VERIFICACIÓN, mientras que las pruebas de aceptación son pruebas de VALIDACIÓN. En ambos casos nuestro SUT estará formado por todo el código de nuestra aplicación.
- Las pruebas del sistema se diseñan desde el punto de vista del desarollador mientras que las pruebas de aceptación se diseñan desde el punto de vista del usuario final. En ambos casos se deben tener en cuenta varios entradas y resultados esperados "intermedios".
- Tanto las pruebas del sistema como las de aceptación se basan en las propiedades emergentes.. Dadas dichas propiedades emergentes, el objetivo de las pruebas del sistema es encontrar defectos en las funcionalidades de dicho sistema, mientras que el objetivo de las pruebas de aceptación es comprobar que se satisfacen los criterios de aceptación.

DISEÑO DE PRUEBAS DE ACEPTACIÓN

- En ambos casos se usan técnicas de caja negra.
- En una prueba del sistema la selección de comportamientos a probar se hace desde un punto de vista técnico (se tienen en cuenta cómo se ha implementado los componentes implicados en las funcionalidades probadas).
- En una prueba de aceptación el diseño se realiza pensando siempre en el uso "real" de nuestra aplicación por el usuario o usuarios finales. (no tenemos en cuenta qué componentes están implicados).

AUTOMATIZACIÓN DE LAS PRUEBAS DE ACEPTACIÓN CON SELENIUM IDE

- Seleccionamos diferentes escenarios de uso de nuestra aplicación. Selenium IDE NO es un lenguaje de programación, es una herramienta que nos permite generar scripts (de forma automática o manual), que podemos ejecutar desde un navegador, por lo tanto podremos usarla si nuestra aplicación tiene una interfaz web.
- Aunque Selenium IDE no es un lenguaje de programación, en las últimas versiones se han incorporado comandos para controlar el flujo de ejecución de nuestros scripts (incluyendo por ejemplo acciones condicionales y bucles).
- No es posible integrar Selenium IDE en una herramienta de construcción de proyectos, lo que constituye una limitación frente a otras aproximaciones.
- En los ejercicios propuestos se han trabajado una serie de comandos, que es necesario conocer y saber aplicar.
- Recuerda que esta práctica, aunque es muy guiada y se os proporciona la solución, requiere un trabajo personal para poder asimilar todos los conceptos trabajados en clase.